[illegible]

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/862,654	05/22/2001	Nigel Peter Topham	0808.65559	8723

24978 7590 03/25/2004

GREER, BURNS & CRAIN
300 S WACKER DR
25TH FLOOR
CHICAGO, IL 60606

EXAMINER	
GOLE, AMOL V	

ART UNIT	PAPER NUMBER
2183	5

DATE MAILED: 03/25/2004

Please find below and/or attached an Office communication concerning this application or proceeding.

PTO-90C (Rev. 10/03)

[illegible]

Office Action Summary	Application No.	Applicant(s)
	09/862,654	TOPHAM, NIGEL PETER
	Examiner	Art Unit
	Amol V. Gole	2183

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on 5/22/01,1/31/02.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) 1-37 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) 1-37 is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on 22 May 2001 is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☒ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☒ All b)☐ Some * c)☐ None of:

1.☒ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☒ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date 2.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application (PTO-152)

6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-37 have been examined.

Papers Submitted

2. It is hereby acknowledged that the following papers have been received and placed of record in the file:

#2: IDS (5/22/01)

#3: Priority Paper (5/22/01)

#4: Change of Address (1/31/02)

Specification

3. The abstract of the disclosure is objected to because it contains more than a 150 words. Correction is required. See MPEP § 608.01(b).

4. The disclosure is objected to because of the following informalities: The various section headings of the disclosure are underlined. Please un-underline these section headings.

5. The lengthy specification has not been checked to the extent necessary to determine the presence of all possible minor errors. Applicant's cooperation is requested in correcting any errors of which applicant may become aware in the specification.

6. The title of the invention is not descriptive. A new title is required that is clearly indicative of the invention to which the claims are directed.

The following title is suggested: PROCESSOR AND METHOD FOR GENERATING AND STORING COMPRESSED INSTRUCTIONS IN A PROGRAM MEMORY AND DECOMPRESSED INSTRUCTIONS IN AN INSTRUCTION CACHE WHEREIN THE DECOMPRESSED INSTRUCTIONS ARE ASSIGNED IMAGINARY ADDRESSES DERIVED FROM INFORMATION STORED IN THE PROGRAM MEMORY WITH THE COMPRESSED INSTRUCTIONS.

Appropriate correction is required.

Claim Rejections - 35 USC § 112

7. The following is a quotation of the second paragraph of 35 U.S.C. 112:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

8. Claims **6, 7, 8, 15, 16, and 19** are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

9. Claim **6** recites the limitation "said one section" in pg.36, line 33. There is insufficient antecedent basis for this limitation in the claim. However for the purposes of examining, this limitation will be interpreted as “said at least one section”.

10. Claim **7** recites the limitation "the section" in pg. 37, line 8. There is insufficient antecedent basis for this limitation in the claim. However for the purposes of examining, this limitation will be interpreted as “the at least one section”.

11. Claim **15** recites the limitation "the stored information" in pg. 38, lines15-16. There is insufficient antecedent basis for this limitation in the claim. However for the purposes of examining, this limitation will be interpreted as “the stored next-section-locating information”.

12. Claim **15** recites the limitation "the cache block" in pg. 38, lines 18-19. There is insufficient antecedent basis for this limitation in the claim. However for the purposes of examining, this limitation will be interpreted as “the most-recently-accessed cache block”.

13. Claims **8, 16, and 19** are rejected as being dependant on claims rejected under 35 U.S.C. 112, second paragraph.

Claim Rejections - 35 USC § 102

13. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

14. Claims **1, 2, 5, 10-18, 20, 23, and 37** are rejected under 35 U.S.C. 102(b) as being anticipated by Faraboschi et al. (US005870576A).

15. In regard to claim 1:

16. Faraboschi et al. teach a processor, for executing instructions of a program stored in compressed form in a program memory (fig. 3, main memory 110; col.4, lines 45-46), comprising:

a program counter which identifies a position in said program memory (code pointer section 152 contains an offset which is used to address a compressed instruction in the memory 110 [col. 6, lines18-20]. Also col. 6, lines 24-27, 39-40 indicate that the cache refill state machine 204 contains a program counter for addressing compressed instructions in the memory 110 which is incremented every time the next instruction in the compressed memory is to be accessed; fig. 4, blocks 306 and 314.

Hence the program counter for identifying a position in the memory 110 is deemed inherent);

an instruction cache (fig. 3, 100), having a plurality of cache blocks (fig. 2, shows that the instruction cache 100 with a plurality of cache blocks), each for storing one or more instructions of said program in decompressed form (fig. 2; col. 4, lines 5-8; col. 5, lines 29-32);

a cache loading unit (fig. 3, cache refill state machine 204, expansion logic 210, and refill buffer 212), comprising a decompression section (fig. 3, expansion logic 210 and refill buffer 212), operable to perform a cache loading operation in which one or more compressed-form instructions are read from said position in the program memory identified by the program counter and are decompressed and stored in one of said cache blocks of the instruction cache (col. 5, lines 24-34);

a cache pointer which identifies a position in said instruction cache of an instruction to be fetched for execution (program counter 200; col. 5, lines 19-21);

an instruction fetching unit which fetches an instruction to be executed from the position identified by the cache pointer (program counter 200) and which, when a cache miss occurs because the instruction to be fetched is not present in the instruction cache, causes the cache loading unit to perform said cache loading operation (fig. 3 shows that the instruction buffer 220 and comparator 208 comprise a fetching unit because the instruction buffer 220 fetches the instruction from the instruction cache based on the cache pointer (program counter 200) address as shown and the comparator compares the address with the cache tags to determine if there was a cache miss or not and further communicates this information to the cache loading unit to perform the cache loading operation [col. 6, lines 24-29]);

and an updating unit (fig. 3, 206 and input line from execution units 230) which updates the program counter (program counter inherent in cache refill state machine 204) and cache pointer (program counter 200) in response to the fetching of instructions so as to ensure that said position identified by said program counter is maintained consistently at the position in said program memory at which the instruction to be fetched from the instruction cache is stored in compressed form (On fetching instructions, the updating unit updates the cache pointer (program counter 200) to point to the next instruction to fetch as can be seen from the fig. 3. Also, the cache pointer (program counter 200) also addresses the appropriate code pointer 152 in the memory 110 [col. 4, lines 53-55], which is used to generate the inherent program counter in the cache refill state machine 204 that addresses the compressed form of the instruction. Hence the updating unit also updates the inherent program counter instantly while updating the cache pointer (program counter 200)).

17. In regard to claim 2:

18. Faraboschi et al. disclose a processor as claimed in claim 1, wherein said position in the instruction cache of an instruction to be fetched is identified by said cache pointer (program counter 200) in terms of an imaginary address (successive instruction addresses col. 5, lines 19-21) assigned to the instruction, at which the

instruction is considered to exist when held in decompressed form in one of said cache blocks (fig. 2; col. 4, lines 5-8; col. 5, lines 29-32).

19. In regard to claim 5:

20. A processor as claimed in claim 1, wherein:

the compressed-form instructions are stored in the program memory in one or more compressed sections (fig. 2, multiple sections comprising compressed instructions and a corresponding mask 150 and code pointer 152 are shown), the compressed-form instructions belonging to each section occupying one of said cache blocks when decompressed (col. 5, lines 29-32; instruction memory has cache blocks), and at least one section also contains imaginary address information relating to the instructions belonging to the section (fig. 2, imaginary address information is stored in code pointer 152); and

said cache loading unit is operable, in said cache loading operation (cache refill state machine 204, expansion logic 210, and refill buffer 212 perform the cache loading operation), to decompress and load into one of said cache blocks one such compressed section stored at the position in the program memory identified by the program counter (col. 6, lines 19-35).

21. In regard to claim 10:

22. Faraboschi et al. disclose a processor as claimed in claim 5, wherein each said compressed section contains imaginary address information (fig. 2, code pointer 152) relating to the instructions belonging to the section concerned (fig. 2 shows code pointer 152 contains imaginary address information relating to the compressed instructions belonging to the section concerned).

23. In regard to claim 11:

24. Faraboschi et al. disclose a processor as claimed in claim 5, wherein the or each said compressed section further contains a decompression key (fig. 2, mask 150) which is employed by said decompression section (expansion logic 210) to effect the decompression of the instructions belonging to the compressed section during the cache loading operation (col. 6, lines 29-34).

25. In regard to claim 12:

26. Faraboschi et al. disclose a processor as claimed in claim 11, wherein the instructions of said program include, prior to compression, preselected instructions (nop instructions indicated by the empty locations in cache 100 of fig. 2) that are not stored explicitly in any said compressed section (fig .2 shows that the nop instructions are not stored in the compressed section 140 of the memory 110), and the decompression key (fig. 2, mask 150) of the or each compressed section identifies the positions at which the preselected instructions are to appear in the cache block when the compressed

section is decompressed (col. 5, lines 4-13).

27. In regard to claim 13:

28. Faraboschi et al. disclose processor as claimed in claim 12, wherein said preselected instructions are "no operation" instructions (fig .2 shows that the nop instructions are not stored in the compressed section 140 of the memory 110).

29. In regard to claim 14:

30. A processor as claimed in claim 5, wherein said updating unit comprises:

a next-section locating section (program counter 200, mask 150, and code pointer 152) operable, in the event of such a cache miss, to employ next-section-locating information (address in program counter 200, mask information 150, and code pointer 152 address), stored in association with the cache block which was accessed most recently to fetch an instruction (address in program counter is also stored in the cache tag corresponding to the cache block as indicated in fig. 2 and col. 5, lines 48-50; as shown in fig. 2, there is a mask 150 and a code pointer stored in association with each cache block in the memory 110), to locate the position in the program memory of a next compressed section following the compressed section corresponding to that most-recently-accessed cache block (On a cache miss, the instruction address in the program counter 200 is used to access the code pointer segment 130 in the memory 110 [col. 6, lines 11, 16]. The code pointer 152 is then used to locate the position in the memory 110 of the next compressed section following the compressed section corresponding to that most-recently-accessed cache block [col. 6, lines 16-24]. The position located in the compressed memory is the *next* compressed section because compressed sections are arranged in consecutive memory locations [col. 6, lines 20-24]).

31. In regard to claim 15:

32. Faraboschi et al. disclose a processor as claimed in claim 14, wherein such next-section-locating information is stored in association with each cache block in which valid decompressed instructions are held (fig. 2 indicates that the information is stored in the cache-tag, mask, and code pointer association with the cache block in which there are valid instructions), the stored information being for use in locating the position in the program memory of the next compressed section following the compressed section corresponding to the cache block concerned (On a cache miss, the instruction address in the program counter is used to access the code pointer segment 130 in the memory 110 [col. 6, lines 11, 16]. The code pointer is then used to locate the position in the memory 110 of the next compressed section following the compressed section corresponding to that most-recently-accessed cache block [col. 6, lines 16-24]. The position located in the compressed memory is the *next* compressed section because compressed sections are arranged in consecutive memory locations [col. 6, lines 20-24]).

33. In regard to claim 16:

34. A processor as claimed in claim 15, wherein the next-section-locating information is stored in association with each cache block when that block is loaded in such a cache loading operation (Although not explicitly mentioned, the limitation is deemed inherent. col. 6, lines 11-53 indicate the cache loading operation. The information in the cache tag 120 must contain the address of the cache block with which it is associated so that it can be determined whether there was a cache hit or miss while comparing it with the instruction address [fig. 3 comparator 208, col. 5, lines 48-50]. So once the cache is loaded with the new block on a cache miss, the new address information must also be stored in the cache tag for proper working of the system).

35. In regard to claim 17:

36. Faraboschi et al. disclose a processor as claimed in claim 14, wherein said next-section-locating information associated with the cache block relates to a size of the compressed section corresponding to that cache block (mask 150 indicates the number of operations in the wide instruction word, i.e. the size of the compressed section, which is to be stored in the cache block [col. 6, lines 37-39]).

37. In regard to claim 18:

38. A processor as claimed in claim 17, wherein said size is determined by the cache loading unit when loading the cache block in the cache loading operation (col. 6, lines 27-39 indicates that the size is determined during the cache loading operation by the cache refill unit because it uses the mask to find out how many instructions are to be loaded into the instruction cache 100).

39. In regard to claim 20:

40. A processor as claimed in claim 12, wherein:

said updating unit comprises:

a next-section locating section (program counter 200, mask 150, and code pointer 152) operable, in the event of such a cache miss, to employ next-section-locating information (address in program counter 200, mask information 150, and code pointer 152), stored in association with the cache block which was accessed most recently to fetch an instruction (address in program counter is also stored in the cache tag corresponding to the cache block as indicated in fig. 2 and col. 5, lines 48-50; as shown in fig. 2, there is a mask 150 and a code pointer stored in association with each cache block in the memory 110), to locate the position in the program memory of a next compressed section following the compressed section corresponding to that most-recently-accessed cache block (On a cache miss, the instruction address in the program counter is used to access the code pointer segment 130 in the memory 110 [col. 6, lines 11, 16]. The code pointer is then used to locate the position in the memory

110 of the next compressed section following the compressed section corresponding to that most-recently-accessed cache block [col. 6, lines 16-24]. The position located in the compressed memory is the *next* compressed section because compressed sections are arranged in consecutive memory locations [col. 6, lines 20-24]); and

said next-section-locating information associated with the cache block represents the number of instructions held in that cache block that are not said preselected instructions (mask 150 indicates the number of operations in the wide instruction word which are to be stored in the cache block [col. 5, lines 5-7; col. 6, lines 37-39]; the preselected instructions are no-operations [fig .2 shows that the nop instructions are not stored in the compressed section 140 of the memory 110]).

41. In regard to claim 23:

42. Faraboschi et al. disclose a processor as claimed in claim 1, wherein the instructions of said program comprise very-long-instruction-word (VLIW) instructions (col. 4, lines 5-21).

43. In regard to claim 37:

44. Faraboschi et al. teach a processor, for executing instructions of a program stored in compressed form in a program memory (fig. 3, main memory 110; col.4, lines 45-46), comprising:

a program counter which identifies a position in said program memory (code pointer section 152 contains an offset which is used to address a compressed instruction in the memory 110 [col. 6, lines18-20]. Also col. 6, lines 24-27, 39-40 indicate that the cache refill state machine 204 contains a program counter for addressing compressed instructions in the memory 110 which is incremented every time the next instruction in the compressed memory is to be accessed; fig. 4, blocks 306 and 314. Hence the program counter for identifying a position in the memory 110 is deemed inherent);

an instruction cache (fig. 3, 100), having a plurality of cache blocks (fig. 2, shows that the instruction cache 100 with a plurality of cache blocks), each for storing one or more instructions of said program in decompressed form (fig. 2; col. 4, lines 5-8; col. 5, lines 29-32);

cache loading means (fig. 3, cache refill state machine 204, expansion logic 210, and refill buffer 212), including decompression means (fig. 3, expansion logic 210 and refill buffer 212), operable to perform a cache loading operation in which one or more compressed-form instructions are read from said position in the program memory identified by the program counter and are decompressed and stored in one of said cache blocks of the instruction cache (col. 5, lines 24-34);

a cache pointer which identifies a position in said instruction cache of an instruction to be fetched for execution (program counter 200; col. 5, lines 19-21);

instruction fetching means for fetching an instruction to be executed from the position identified by the cache pointer (program counter 200) and which, when a cache miss occurs because the instruction to be fetched is not present in the instruction cache, causes the cache loading unit to perform said cache loading operation (fig. 3 shows that the instruction buffer 220 and comparator 208 comprise a fetching unit because the instruction buffer 220 fetches the instruction from the instruction cache based on the cache pointer (program counter 200) address as shown and the comparator compares the address with the cache tags to determine if there was a cache miss or not and further communicates this information to the cache loading unit to perform the cache loading operation [col. 6, lines 24-29]);

and updating means (fig. 3, 206 and input line from execution units 230) for updating the program counter (program counter inherent in cache refill state machine 204) and cache pointer (program counter 200) in response to the fetching of instructions so as to ensure that said position identified by said program counter is maintained consistently at the position in said program memory at which the instruction to be fetched from the instruction cache is stored in compressed form (On fetching instructions, the updating unit updates the cache pointer (program counter 200) to point to the next instruction to fetch as can be seen from the fig. 3. Also, the cache pointer (program counter 200) also addresses the appropriate code pointer 152 in the memory 110 [col. 4, lines 53-55], which is used to generate the inherent program counter in the cache refill state machine 204 that addresses the compressed form of the instruction. Hence the updating unit also updates the inherent program counter instantly while updating the cache pointer (program counter 200)).

Claim Rejections - 35 USC § 103

45. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

46. Claims **3, 4, 6-9, 19, 21, and 27-35** are rejected under 35 U.S.C. 103(a) as being unpatentable over Faraboschi et al. (US005870576A).

47. In regard to claim 3:

48. While Faraboschi et al. do teach imaginary addresses (successive instruction addresses col. 5, lines 19-21), they do not expressly disclose a processor as claimed in claim 2, wherein said imaginary address of an instruction is assigned thereto during assembly/linking of said program based on the sequence of original instructions in the program prior to compression.

49. However, “Official Notice” is taken that it is well known and expected in the art to use a linker to assign addresses to a program. This is substantiated by the definition of a linker provided by http://www.webopedia.com/TERM/l/linker.html (pg. 2, lines 14-17).

50. Therefore it would have been obvious for one of ordinary skill in the art at the time of the invention to have recognized that the imaginary addresses (successive instruction addresses col. 5, lines 19-21) must be assigned during the linking of the program based on the original sequence prior to compression.

51. One would have been motivated to do so because it is well known that addresses are assigned during linking.

52. In regard to claim 4:

53. Faraboschi et al. do not disclose that the imaginary address information, from which said imaginary address assigned to each instruction is derivable, is stored with the compressed-form instructions in the program memory and is employed in the cache loading operation so as to associate with each decompressed instruction present in the instruction cache the imaginary address assigned thereto.

54. However “Official Notice” is taken that it is well known that a program counter stores address information that points to the address of instruction to be accessed in the instruction memory (fig. 2 shows that lines in the code pointer segment having addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag.

55. One of ordinary skill in the art at the time of the invention would have recognized that by storing the address information along with the compressed instructions in the memory 110, it would allow for faster lookups and enable easy setting of the cache tag when loading the cache with instructions from the memory 110. Therefore it would have been obvious to one of ordinary skill in the art at the time of the invention to store the imaginary address information along with the compressed instructions.

56. One would have been motivated to do so because it allows for faster lookups and hence shorter clock period and simpler circuitry.

57. In regard to claim 6:

58. Faraboschi et al. do not disclose that the imaginary address information of said one section specifies the imaginary address at which a first one of the decompressed instructions corresponding to the compressed section is considered to exist when the decompressed instructions are held in one of the cache blocks.

59. However "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of instruction to be accessed in the instruction memory (fig. 2 shows that lines in the code pointer segment having addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag.

60. One of ordinary skill in the art at the time of the invention would have recognized that by storing the address information along with the compressed instructions in the memory 110, it would allow for faster lookups and enable easy setting of the cache tag

when loading the cache with instructions from the memory 110. This information would specify the imaginary address of the first one of the decompressed instructions because the program counter 200 points to the cache block where the instruction is to be stored and the location in memory 110 where the instruction information is stored. Therefore it would have been obvious to one of ordinary skill in the art at the time of the invention to store the imaginary address information along with the section in the memory 110.

61. One would have been motivated to do so because it allows for faster lookups and hence shorter clock period and simpler circuitry.

62. In regard to claim 7:

63. Faraboschi et al. do not disclose that in the cache loading operation the cache block into which the decompressed instructions of the compressed section are loaded is assigned an imaginary block address based on said imaginary address assigned to an instruction contained in the section.

64. However "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of instruction to be accessed in the instruction memory (fig. 2 shows that lines in the code pointer segment having addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag.

65. One of ordinary skill in the art at the time of the invention would have recognized that by storing the address information along with the compressed instructions in the memory 110, it would allow for faster lookups and enable easy setting of the cache tag

when loading the cache with instructions from the memory 110. This information assigns the imaginary address of the decompressed instruction because the program counter 200 points to the cache block where the instruction is to be stored and the location in memory 110 where the instruction information is stored. Therefore it would have been obvious to one of ordinary skill in the art at the time of the invention to store the imaginary address information along with the section in the memory 110 and use it to assign the cache tag and therefore address of the cache block of the decompressed instruction.

66. One would have been motivated to do so because it allows for faster lookups and hence shorter clock period and simpler circuitry.

67. In regard to claim 8:

68. Faraboschi et al. discloses a processor as claimed in claim 7, wherein each said cache block has an associated cache tag in which is stored said imaginary block address assigned to the cache block with which the cache tag is associated (fig. 2, 120 and col. 5, lines 48-50).

69. In regard claim 9:

70. Faraboschi et al. do not disclose that imaginary address information is contained in only a first one of said compressed sections to be loaded.

71. However "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of instruction to be accessed in the

instruction memory (fig. 2 shows that lines in the code pointer segment having addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag. However, in order to save space, only the location of the first entry of the section can be saved from which the other sections can be addressed using relative addressing.

72. One of ordinary skill in the art at the time of the invention would have recognized that by storing the address information along with the first one of the compressed sections in the memory 110, it would allow for faster lookups, savings in space as compared to storing the address information in all the sections, and enable easy setting of the cache tag when loading the cache with instructions from the memory 110. This information assigns the imaginary address of the decompressed instruction because the program counter 200 points to the cache block where the instruction is to be stored and the location in memory 110 where the instruction information is stored. Therefore it would have been obvious to one of ordinary skill in the art at the time of the invention to store the imaginary address information along with the first one of the sections in the memory 110.

73. One would have been motivated to do so because it allows for faster lookups and hence shorter clock period and simpler circuitry.

74. In regard to claim 19:

75. Faraboschi et al. discloses that a processor as claimed in claim 15, wherein the updating unit comprises:

a locating information register section (fig. 2, cache tags 120, mask 150, code pointer 152) which stores said next-section-locating information associated with the most-recently-accessed cache block;

said next-section-locating section being operable, in the event of such a cache miss, to employ the next-section-locating information stored in the location information register section to locate said position of said next compressed section (On a cache miss, the instruction address in the program counter is used to access the code pointer segment 130 in the memory 110 [col. 6, lines 11, 16]. The code pointer is then used to locate the position in the memory 110 of the next compressed section following the compressed section corresponding to that most-recently-accessed cache block [col. 6, lines 16-24]. The position located in the compressed memory is the *next* compressed section because compressed sections are arranged in consecutive memory locations [col. 6, lines 20-24]).

76. Faraboschi et al. do not disclose that copying section operable, when an instruction held in one of the cache blocks is fetched, to copy into the locating information register section said next-section-locating information stored in association with that block;

77. However "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of instruction to be accessed in the

instruction memory (fig. 2 shows that lines in the code pointer segment having addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag.

78. One of ordinary skill in the art at the time of the invention would have recognized that by storing the address information along with the first one of the compressed sections in the memory 110, it would allow for faster lookups and enable easy setting of the cache tag when loading the cache with instructions from the memory 110 by copying the information into the cache tag. Therefore it would have been obvious to one of ordinary skill in the art at the time of the invention to store the imaginary address information along with the first one of the sections in the memory 110 and copy the address information into the cache tag in order to set it in a simple fashion.

79. One would have been motivated to do so because it allows for faster lookups and hence shorter clock period and simpler circuitry.

80. In regard to claim 21:

81. Faraboschi et al. do not disclose a processor as claimed in claim 20, wherein the decompression section comprises a counter operable, during such a cache loading operation, to count the number of decompressed instructions that are not said preselected instructions.

82. “Official Notice” is taken that it is well known and expected in the art to use a counter to count a number of instructions to be transferred from one location to another.

83. One of ordinary skill would have recognized that from col. 6, lines 27-39 which indicates that the cache refill unit uses the mask to find out how many decompressed instructions are to be transferred into the instruction cache 100, that a counter to count the number of decompressed instructions that are not no operations could be advantageously used so as to transfer the right amount of instructions to the cache 100.

84. Therefore it would have been obvious to one of ordinary skill in the art the time of the invention to have used a counter to count the number of decompressed instructions that are not said preselected instructions.

85. In regard to claim 27:

86. Faraboschi et al. disclose a method of compressing a program to be executed by a processor in which compressed-form instructions stored in a program memory (fig. 3, main memory 110; col. 4, lines 45-46) are decompressed and cached in an instruction cache prior to being issued (col. 5, lines 29-32), the method comprising:

converting a sequence of original instructions of the program into a corresponding sequence of such compressed-form instructions (Although not explicitly mentioned, this limitation is deemed inherent as fig. 2 shows compressed instructions stored in a sequence and they must be converted from a sequence of original instructions of the program);

assigning such original instructions imaginary addresses according to said sequence thereof, the assigned imaginary addresses being imaginary addresses at which the instructions are to be considered to exist when held in decompressed form in said instruction cache of the processor (Although not explicitly mentioned, this limitation is deemed inherent as during the compiling and storing of the program, the original instructions must be assigned instructions. As the decompressed form of instructions are the same as the original form, the addresses will be the same);

87. However, Faraboschi et al. do not disclose the step of storing, in said program memory, the compressed-form instructions together with imaginary address information specifying said assigned imaginary addresses so that, when the compressed-form instructions are decompressed and loaded by the processor into the instruction cache, the processor can assign the specified imaginary addresses to the decompressed instructions.

88. However "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of instruction to be accessed in the instruction memory (fig. 2 shows that lines in the code pointer segment having addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag.

89. One of ordinary skill in the art at the time of the invention would have recognized that by storing the address information along with the first one of the compressed sections in the memory 110, it would allow for faster lookups and enable easy setting of the cache tag when loading the cache with instructions from the memory 110 by copying the information into the cache tag. This information assigns the imaginary address of the decompressed instruction because the program counter 200 points to the cache block where the instruction is to be stored and the location in memory 110

where the instruction information is stored. Therefore it would have been obvious to one of ordinary skill in the art at the time of the invention to store the imaginary address information along with the section in the memory 110 and use it to assign the cache tag and therefore address of the cache block of the decompressed instruction.

90. One would have been motivated to do so because it allows for faster lookups and hence shorter clock period and simpler circuitry.

91. In regard to claim 28:

92. Faraboschi et al. disclose a method as claimed in claim 27, wherein the assigned imaginary addresses are selected so that instructions likely to coexist in the instruction cache at execution time will not be mapped to the same cache block (fig. 2, discloses a direct mapped cache and therefore the addresses assigned will likely not cause the instructions to be mapped to the same cache block).

93. In regard to claim 29:

94. Faraboshi et al. disclose a method as claimed in claim 27, wherein the compressed-form instructions are stored in the program memory in one or more compressed sections (fig. 2, multiple sections comprising compressed instructions and a corresponding mask 150 and code pointer 152 are shown), the compressed-form instructions belonging to each section occupying one cache block of the processor's instruction cache when decompressed (col. 5, lines 29-32; instruction memory has cache blocks), and at least one compressed section also containing imaginary address

information relating to the instructions belonging to the section (fig. 2, imaginary address information is stored in code pointer 152); and

95. In regard to claim 30:

96. Faraboschi et al. do not disclose that the imaginary address information specifies the imaginary address at which a first one of the decompressed instructions corresponding to the compressed section is considered to exist when the decompressed instructions are held in said instruction cache.

97. However "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of instruction to be accessed in the instruction memory (fig. 2 shows that lines in the code pointer segment having addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag.

98. One of ordinary skill in the art at the time of the invention would have recognized that by storing the address information along with the compressed instructions in the memory 110, it would allow for faster lookups and enable easy setting of the cache tag when loading the cache with instructions from the memory 110. This information would specify the imaginary address of the first one of the decompressed instructions because the program counter 200 points to the cache block where the instruction is to be stored and the location in memory 110 where the instruction information is stored. Therefore it would have been obvious to one of ordinary skill in the art at the time of the invention to store the imaginary address information along with the section in the memory 110.

99. One would have been motivated to do so because it allows for faster lookups and hence shorter clock period and simpler circuitry.

100. In regard to claim 31:

101. Faraboschi et al. do not disclose that imaginary address information is contained in only a first one of said compressed sections to be loaded.

102. However "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of instruction to be accessed in the instruction memory (fig. 2 shows that lines in the code pointer segment having addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag. However, in order to save space, only the location of the first entry of the section can be saved from which the other sections can be addressed using relative addressing.

103. One of ordinary skill in the art at the time of the invention would have recognized that by storing the address information along with the first one of the compressed sections in the memory 110, it would allow for faster lookups, savings in space as compared to storing the address information in all the sections, and enable easy setting of the cache tag when loading the cache with instructions from the memory 110. This information assigns the imaginary address of the decompressed instruction because the program counter 200 points to the cache block where the instruction is to be stored and the location in memory 110 where the instruction information is stored. Therefore it would have been obvious to one of ordinary skill in the art at the time of the invention to

store the imaginary address information along with the first one of the sections in the memory 110.

104. One would have been motivated to do so because it allows for faster lookups and hence shorter clock period and simpler circuitry.

105. In regard to claim 32:

106. Faraboschi et al. disclose a method as claimed in claim 29, wherein each said compressed section contains imaginary address information (fig. 2, code pointer 152) relating to the instructions belonging to the section concerned (fig. 2 shows code pointer 152 contains imaginary address information relating to the compressed instructions belonging to the section concerned).

107. In regard to claim 33:

108. Faraboschi et al. disclose a method as claimed in claim 29, wherein the or each said compressed section further contains a decompression key (fig. 2, mask 150) for use by the processor to carry out the decompression of the instructions belonging to the compressed section during the cache loading operation (col. 6, lines 29-34).

109. In regard to claim 34:

110. Faraboschi et al. disclose a processor as claimed in claim 33, wherein said sequence of original instructions of the program comprises preselected instructions (nop instructions indicated by the empty locations in cache 100 of fig. 2) that are not stored

explicitly in any said compressed section (fig .2 shows that the nop instructions are not stored in the compressed section 140 of the memory 110), and the decompression key (fig. 2, mask 150) of the or each compressed section identifies the positions at which the preselected instructions exist are to appear in a decompressed sequence of instructions corresponding to the section (col. 5, lines 4-13).

111. In regard to claim 35:

112. Faraboschi et al. disclose a method as claimed in claim 34, wherein said preselected instructions are "no operation" instructions (fig .2 shows that the no operation (nop) instructions are not stored in the compressed section 140 of the memory 110).

113. Claims **22, 24-26** are rejected under 35 U.S.C. 103(a) as being unpatentable over Faraboschi et al. (US005870576A) in view of Guttag et al. (US005509129A).

114. In regard to claim 22:

115. Faraboschi et al. do not disclose a processor as claimed in claim 1, operable to execute a hardware-controlled loop, wherein:

said updating unit further comprises respective first and second loop control registers and operates, upon initiation of execution of such a hardware-controlled loop, to cause the program-counter value to be stored in said first loop control register and to cause the cache-pointer value to be stored in said second loop control register, and further operates, upon commencement of each iteration of the loop after said first iteration thereof, to reload said program counter with the value held in said first loop control register and to reload said cache pointer with the value held in said second loop control register.

116. However Guttag et al. disclose program flow control unit in a long instruction word processor as shown in fig. 31 where the program counter 701 is connected to a loop control register (loop start register LS1). Col. 162, lines 13+, col. 163, lines 1-50 disclose the operation of the loop logic. Col. 162, lines 41-44 disclose that during the initialization of the hardware controlled loop, the program counter value is stored in the loop control register (LS1) and on commencement of each iteration of the loop (loop count register equals zero col. 163, lines 7-8), the program counter is reloaded with the loop control register (LS1) value (col. 163, lines 7-14). Thus Guttag et al. detail a zero-overhead loop logic to control hardware branching (col. 162, lines 12-14)

117. One of ordinary skill would have easily recognized that the zero-overhead loop logic proposed by Guttag et al. for a VLIW processor could be advantageously used in the Faraboschi et al. VLIW processor to execute loops quickly. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to have modified the update unit to have loop control registers for each of the program counter and cache pointer (as they both serve the same purpose of the program counter of Guttag et al.) and to load the program counter and cache pointer values into their respective loop control register upon initiation of the loop and reload the program

counter and cache pointer with the values in their respective loop control register on completion of an iteration of the loop as taught by Guttag et al.

118. It would have been obvious to do so because the addition of the loop logic leads to zero-overhead execution of loops which would improve the performance of the Faraboschi et al. processor.

119. In regard to claim 24:

120. Faraboschi et al. do not disclose a processor as claimed in claim 1, wherein the updating unit is operable, when an interrupt occurs during execution of a program, to cause the program-counter value and cache-pointer value to be saved pending handling of the interrupt, and, when the execution of the program is resumed, to cause the saved values to be restored in the program counter and cache pointer.

121. However, Guttag et al. disclose an interrupt handling mechanism (fig. 31, 706, 707, 770, 702, 703, 701 and 704) wherein when an enabled interrupt occurs and the interrupt service routine branches to another location, the program counter 701 value is stored in an instruction pointer-return from subroutine register (IPRS 704) in order to be able to use that register value to return from the interrupt service routine by restoring the program counter contents (col. 102, 35-56; col. 110, 28-30, 50-56).

122. One of ordinary skill would have easily recognized that the interrupt handling technique proposed by Guttag et al. for a VLIW processor could be advantageously used in the Faraboschi et al. VLIW processor to enable the handling of interrupt requests and allow for I/O interrupt handling, exception handling, and other well known

useful services in the art. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to have modified the update unit to have an interrupt handling mechanism to save the values of the program counter and cache pointer (as they both serve the same purpose of the program counter of Guttag et al.) in a IPRS register on an interrupt and further using those saved values to restore the contents of the program counter and cache pointer to resume execution of the program.

123. It would have been obvious to do so because the addition of the interrupt handling mechanism would enable the handling of interrupt requests and increase the functionality of the VLIW processor.

124. In regard to claim 25:

125. Faraboschi et al. do not disclose a processor as claimed in claim 14, wherein the updating unit is operable, when an interrupt occurs during execution of a program, to cause said next-section locating information associated with the most-recently-accessed cache block to be saved pending handling of the interrupt, and, when execution of the program is resumed, to cause the saved next-section locating information to be restored.

126. However, Guttag et al. disclose an interrupt handling mechanism (fig. 31, 706, 707, 770, 702, 703, 701 and 704) wherein when an enabled interrupt occurs and the interrupt service routine branches to another location, the program counter 701 value (which is the next-section locating information) is stored in an instruction pointer-return from subroutine register (IPRS 704) in order to be able to use that register value to

return from the interrupt service routine by restoring the program counter contents (col. 102, 35-56; col. 110, 28-30, 50-56).

127. One of ordinary skill would have easily recognized that the interrupt handling technique proposed by Guttag et al. for a VLIW processor could be advantageously used in the Faraboschi et al. VLIW processor to enable the handling of interrupt requests and allow for I/O interrupt handling, exception handling, and other well known useful services in the art. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to have modified the update unit to have an interrupt handling mechanism to save the values of the next-section locating information i.e. the program counter 200 and cache pointer 152 (as they both serve the same purpose of the program counter of Guttag et al.) on an interrupt and further using those saved values to restore the contents of the program counter and cache pointer to resume execution of the program.

128. It would have been obvious to do so because the addition of the interrupt handling mechanism would enable the handling of interrupt requests and increase the functionality of the VLIW processor.

129. In regard to claim 26:

130. Faraboschi et al. do not disclose a processor as claimed in claim 20, wherein the updating unit is operable, when an interrupt occurs during execution of a program, to cause the values held in said loop control registers to be saved pending handling of the

interrupt, and, when execution of the program is resumed, to cause the saved values to be restored in the loop control registers.

131. However, Guttag et al. disclose an interrupt handling mechanism (fig. 31, 706, 707, 770, 702, 703, 701 and 704) wherein when an enabled interrupt occurs and the interrupt service routine branches to another location, the program counter 701 value (which is the value stored in the loop control registers LS 721-723) is stored in an instruction pointer-return from subroutine register (IPRS 704) in order to be able to use that register value to return from the interrupt service routine by restoring the program counter contents (col. 102, 35-56; col. 110, 28-30, 50-56).

132. One of ordinary skill would have easily recognized that the interrupt handling technique proposed by Guttag et al. for a VLIW processor could be advantageously used in the Faraboschi et al. VLIW processor to enable the handling of interrupt requests and allow for I/O interrupt handling, exception handling, and other well known useful services in the art. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to have modified the update unit to have an interrupt handling mechanism to save the values of the loop control registers which store the program counter 200 and cache pointer 152 values on an interrupt and further using those saved values to restore the contents of the loop control registers to resume execution of the program.

133. It would have been obvious to do so because the addition of the interrupt handling mechanism would enable the handling of interrupt requests and increase the functionality of the VLIW processor.

134. Claim **36** is rejected under 35 U.S.C. 103(a) as being unpatentable over Faraboschi et al. (US005870576A) in view of in view of Tannenbaum ("Structured Computer Organization," Prentice-Hall, 1984, pp. 10-12).

135. In regard to claim 36:

136. Faraboschi et al. disclose a method of compressing a program to be executed by a processor in which compressed-form instructions stored in a program memory (fig. 3, main memory 110; col. 4, lines 45-46) are decompressed and cached in an instruction cache prior to being issued (col. 5, lines 29-32), the method comprising:

converting a sequence of original instructions of the program into a corresponding sequence of such compressed-form instructions (Although not explicitly mentioned, this limitation is deemed inherent as fig. 2 shows compressed instructions stored in a sequence and they must be converted from a sequence of original instructions of the program);

assigning such original instructions imaginary addresses according to said sequence thereof, the assigned imaginary addresses being imaginary addresses at which the instructions are to be considered to exist when held in decompressed form in said instruction cache of the processor (Although not explicitly mentioned, this limitation is deemed inherent as during the compiling and storing of the program, the original instructions must be assigned instructions. As the decompressed form of instructions are the same as the original form, the addresses will be the same);

137. However, Faraboschi et al. do not disclose the step of storing, in said program memory, the compressed-form instructions together with imaginary address information specifying said assigned imaginary addresses so that, when the compressed-form instructions are decompressed and loaded by the processor into the instruction cache, the processor can assign the specified imaginary addresses to the decompressed instructions. Also, Faraboschi et al. differs from the applicant's invention in that it does not teach that instructions on a computer-readable medium storing instructions which carries out the method of the applicant's invention.

138. However "Official Notice" is taken that it is well known that a program counter stores address information that points to the address of instruction to be accessed in the instruction memory (fig. 2 shows that lines in the code pointer segment having addresses) and that address information could easily be stored with the instruction to allow for faster lookups in the memory just like a cache tag.

139. One of ordinary skill in the art at the time of the invention would have recognized that by storing the address information along with the first one of the compressed sections in the memory 110, it would allow for faster lookups and enable easy setting of the cache tag when loading the cache with instructions from the memory 110 by copying the information into the cache tag. This information assigns the imaginary address of the decompressed instruction because the program counter 200 points to the cache block where the instruction is to be stored and the location in memory 110 where the instruction information is stored. Therefore it would have been obvious to one of ordinary skill in the art at the time of the invention to store the imaginary address

information along with the section in the memory 110 and use it to assign the cache tag and therefore address of the cache block of the decompressed instruction.

140. One would have been motivated to do so because it allows for faster lookups and hence shorter clock period and simpler circuitry.

141. Also, Tannenbaum teaches that any instruction executed by hardware can also be simulated in software (pg 11, para. 4, lines 1-2). He also teaches that hardware is generally immutable (first para. after sec. 1.4 header) while software allows for more rapid change (pg. 11, para. 4, lines 2-4).

142. One of ordinary skill in the art at the time of the invention would have been motivated to convert the Smith et al. reference to software i.e. instructions on a machine readable medium because Tannebaum teaches that hardware is generally immutable (first para. after sec. 1.4 header) while software allows for more rapid change (pg. 11, para. 4, lines 2-4). Therefore, to allow for ease of correction of mistakes, and/or an ease of addition of new functionality, one of ordinary skill would be motivated to implement the teachings of Smith et al. and Tanenbaum as instructions recorded on a machine readable medium.

143. Thus, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify the Smith et al. processor by converting it to instructions on a machine-readable medium.

Conclusion

144. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure. Applicant is reminded that in amending in reply to a rejection of claims in an application or patent under reexamination, the applicant or patent owner must clearly point out the patentable novelty, which he or she thinks the claims present in view of the state of the art disclosed by the references cited or the objections made. The applicant or patent owner must also show how the amendments avoid such references or objections. See 37 CFR § 1.111.

a. Vondran, Jr. (US006581131B2) details a main memory storing compressed VLIW instructions and the addresses of the corresponding decompressed instructions.

b. Iverson (US006195107B1) details virtual memory in an embedded system.

c. Colwell et al. (US005057837) details the mask and compressed instructions in fig. 5 and 6.

d. Tananka et al. (US005893143A) also details the mask and compressed instructions in fig. 6.

e. Breternitz, Jr. et al. (US006343354B1) discloses a compression decompression mechanism.

f. Beneveniste et al. (US006349372B1) discloses a compressed main memory.

g. Conte et al. ("Instruction fetch mechanisms for VLIW architectures with compressed encodings", Microarchitecture, 1996. MICRO-29. Proceedings of the 29th Annual IEEE/ACM International Symposium on, Dec. 2, 1996, pp 201-211) teaches fetching mechanisms for VLIW architectures with compressed isntructions.

h. Larin, S.Y.; Conte, T.M.("Compiler-driven cached code compression schemes for embedded ILP processors", Microarchitecture, 1999. MICRO-32. Proceedings, 32nd Annual International Symposium on, Nov. 16, 1999, pp 82-92) teach compression techniques.

145. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Amol V. Gole whose telephone number is 703-305-8888. The examiner can normally be reached on 9:00-6:30.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on 703-305-9712. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

AVG
amol.gole@uspto.gov

[illegible]

EDDIE CHAN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100